Fast computation of power series solutions of systems of differential equations

Alin Bostan, Frédéric Chyzak, François Ollivier, Bruno Salvy, Éric Schost, Alexandre Sedoglavic

FAST COMPUTATION OF POWER SERIES SOLUTIONS OF SYSTEMS OF DIFFERENTIAL EQUATIONS

A. BOSTAN, F. CHYZAK, F. OLLIVIER, B. SALVY, É. SCHOST, AND A. SEDOGLAVIC

Abstract. We propose new algorithms for the computation of the first N terms of a vector (resp. a basis) of power series solutions of a linear system of differential equations at an ordinary point, using a number of arithmetic operations which is quasi-linear with respect to N. Similar results are also given in the non-linear case. This extends previous results obtained by Brent and Kung for scalar differential equations of order one and two.

1. Introduction

In this article, we are interested in the computation of the first N terms of power series solutions of differential equations. This problem arises in combinatorics, where the desired power series is a generating function, as well as in numerical analysis and in particular in control theory.

Let $\mathbb{K}$ be a field. Given $r+1$ formal power series $a_0(t), \ldots, a_r(t)$ in $\mathbb{K}[[t]]$, one of our aims is to provide fast algorithms for solving the linear differential equation

$$a_r(t)y^{(r)}(t) + \cdots + a_1(t)y'(t) + a_0(t)y(t) = 0. \tag{1}$$

Specifically, under the hypothesis that $t = 0$ is an ordinary point for Equation (1) (i.e., $a_r(0) \neq 0$), we give efficient algorithms taking as input the first N terms of the power series $a_0(t), \ldots, a_r(t)$ and answering the following algorithmic questions:

- **i.** find the first N coefficients of the r elements of a basis of power series solutions of (1);
- **ii.** given initial conditions $\alpha_0, \ldots, \alpha_{r-1}$ in $\mathbb{K}$, find the first N coefficients of the unique solution $y(t)$ in $\mathbb{K}[[t]]$ of Equation (1) satisfying

$$y(0) = \alpha_0, \quad y'(0) = \alpha_1, \quad \ldots, \quad y^{(r-1)}(0) = \alpha_{r-1}.$$

More generally, we also treat linear first-order systems of differential equations. From the data of initial conditions v in $\mathcal{M}_{r\times r}(\mathbb{K})$ (resp. $\mathcal{M}_{r\times 1}(\mathbb{K})$) and of the first N coefficients of each entry of the matrices A and B in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ (resp. b in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$), we propose algorithms that compute the first N coefficients:

- **I.** of a fundamental solution Y in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ of $Y' = AY + B$, with $Y(0) = v$, $\det Y(0) \neq 0$;
- **II.** of the unique solution $y(t)$ in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ of $y' = Ay + b$, satisfying $y(0) = v$.

Obviously, if an algorithm of algebraic complexity C (i.e., using C arithmetic operations in $\mathbb{K}$) is available for problem **II**, then applying it r times solves problem **I** in time $r\,\mathsf{C}$, while applying it to a companion matrix solves problem **ii** in time C and problem **i** in $r\,\mathsf{C}$. Conversely, an algorithm solving **i** (resp. **I**) also solves **ii** (resp. **II**) within the same complexity, plus that of a linear combination of series. Our reason for distinguishing the four problems **i, ii, I, II** is that in many cases, we are able to give algorithms of better complexity than obtained by these reductions.

The most popular way of solving **i**, **ii**, **I**, and **II** is the method of undetermined coefficients that requires $\mathcal{O}(r^2N^2)$ operations in $\mathbb{K}$ for problem **i** and $\mathcal{O}(rN^2)$ operations in $\mathbb{K}$ for **ii**. Regarding the dependence in N, this is certainly too expensive compared to the size of the output, which is only linear in N in both cases. On the other hand, verifying the correctness of the output for **ii** (resp. **i**)

Partially supported by a grant from the French *Agence nationale pour la recherche*.

already requires a number of operations in $\mathbb{K}$ which is linear (resp. quadratic) in r: this indicates that there is little hope of improving the dependence in r. Similarly, for problems **I** and **II**, the method of undetermined coefficients requires $\mathcal{O}(N^2)$ multiplications of $r \times r$ scalar matrices (resp. of scalar matrix-vector products in size r), leading to a computational cost which is reasonable with respect to r, but not with respect to N.

By contrast, the algorithms proposed in this article have costs that are linear (up to logarithmic factors) in the complexity $\mathsf{M}(N)$ of polynomial multiplication in degree less than N over $\mathbb{K}$. Using Fast Fourier Transform (FFT) these costs become nearly linear — up to polylogarithmic factors — with respect to N, for all of the four problems above (precise complexity results are stated below). Up to these polylogarithmic terms in N, this estimate is probably not far from the lower algebraic complexity one can expect: indeed, the mere check of the correctness of the output requires, in each case, a computational effort proportional to N.

1.1. **Newton Iteration.** In the case of first-order equations ($r = 1$), Brent and Kung have shown in [8] (see also [15, 23]) that the problems can be solved with complexity $\mathcal{O}(\mathsf{M}(N))$ by means of a formal Newton iteration. Their algorithm is based on the fact that solving the first-order differential equation $y'(t) = a(t)y(t)$, with $a(t)$ in $\mathbb{K}[[t]]$ is equivalent to computing the *power series exponential* $\exp(\int a(t))$. This equivalence is no longer true in the case of a system $Y' = A(t)Y$ (where $A(t)$ is a power series matrix): for non-commutativity reasons, the matrix exponential $Y(t) = \exp(\int A(t))$ is not a solution of $Y' = A(t)Y$.

Brent and Kung suggest a way to extend their result to higher orders, and the corresponding algorithm has been shown by van der Hoeven in [40] to have complexity $\mathcal{O}(r^r\,\mathsf{M}(N))$. This is good with respect to N, but the exponential dependence in the order r is unacceptable.

Instead, we solve this problem by devising a specific Newton iteration for $Y' = A(t)Y$. Thus we solve problems **i** and **I** in $\mathcal{O}(\mathsf{MM}(r, N))$, where $\mathsf{MM}(r, N)$ is the number of operations in $\mathbb{K}$ required to multiply $r \times r$ matrices with polynomial entries of degree less than N. For instance, when $\mathbb{K} = \mathbb{Q}$, this is $\mathcal{O}(r^\omega N + r^2\mathsf{M}(N))$, where r^ω can be seen as an abbreviation for $\mathsf{MM}(r, 1)$, see §1.5 below.

1.2. **Divide-and-conquer.** The resolution of problems **i** and **I** by Newton iteration relies on the fact that a whole basis is computed. Dealing with problems **ii** and **II**, we do not know how to preserve this algorithmic structure, while simultaneously saving a factor r.

To solve problems **ii** and **II**, we therefore propose an alternative algorithm, whose complexity is also nearly linear in N (but not quite as good, being in $\mathcal{O}(\mathsf{M}(N)\log N)$), but whose dependence in the order r is better — linear for **i** and quadratic for **ii**. In a different model of computation with power series, based on the so-called *relaxed multiplication*, van der Hoeven briefly outlines another algorithm [40, Section 4.5.2] solving problem **ii** in $\mathcal{O}(r\,\mathsf{M}(N)\log N)$. To our knowledge, this result cannot be transferred to the usual model of power series multiplication (called zealous in [40]).

We use a divide-and-conquer technique similar to that used in the fast Euclidean algorithm [22, 35, 39]. For instance, to solve problem **ii**, our algorithm divides it into two similar problems of halved size. The key point is that the lowest coefficients of the solution $y(t)$ only depend on the lowest coefficients of the coefficients a_i. Our algorithm first computes the desired solution $y(t)$ at precision only $N/2$, then it recovers the remaining coefficients of $y(t)$ by recursively solving at precision $N/2$ a new differential equation. The main idea of this second algorithm is close to that used for solving first-order difference equations in [13].

We encapsulate our main complexity results in Theorem 1 below. When FFT is used, the functions $\mathsf{M}(N)$ and $\mathsf{MM}(r, N)$ have, up to logarithmic terms, a nearly linear growth in N, see §1.5. Thus, the results in the following theorem are quasi-optimal.

Theorem 1. *Let N and r be two positive integers and let $\mathbb{K}$ be a field of characteristic zero or at least N. Then:*

Problem (input, output)	constant coefficients	polynomial coefficients	power series coefficients	output size
i (equation, basis)	$\mathcal{O}(\mathsf{M}(r)N)$ ⋆	$\mathcal{O}(dr^2N)$	$\mathcal{O}(\mathsf{MM}(r,N))$ ⋆	$\mathcal{O}(rN)$
ii (equation, one solution)	$\mathcal{O}(\mathsf{M}(r)N/r)$ ⋆	$\mathcal{O}(drN)$	$\mathcal{O}(r\,\mathsf{M}(N)\log N)$ ⋆	$\mathcal{O}(N)$
I (system, basis)	$\mathcal{O}(r\mathsf{M}(r)N)$ ⋆	$\mathcal{O}(dr^\omega N)$	$\mathcal{O}(\mathsf{MM}(r,N))$ ⋆	$\mathcal{O}(r^2N)$
II (system, one solution)	$\mathcal{O}(\mathsf{M}(r)N)$ ⋆	$\mathcal{O}(dr^2N)$	$\mathcal{O}(r^2\,\mathsf{M}(N)\log N)$ ⋆	$\mathcal{O}(rN)$

TABLE 1. Complexity of solving linear differential equations/systems for $N \gg r$. Entries marked with a '⋆' correspond to new results.

(a) *problems* **i** *and* **I** *can be solved using* $\mathcal{O}(\mathsf{MM}(r,N))$ *operations in* $\mathbb{K}$*;*
(b) *problem* **ii** *can be solved using* $\mathcal{O}(r\,\mathsf{M}(N)\log N)$ *operations in* $\mathbb{K}$*;*
(c) *problem* **II** *can be solved using* $\mathcal{O}\left(r^2\,\mathsf{M}(N)\log N\right)$ *operations in* $\mathbb{K}$*.*

1.3. **Special Coefficients.** For special classes of coefficients, we give different algorithms of better complexity. We isolate two important classes of equations: that with constant coefficients and that with polynomial coefficients. In the case of constant coefficients, our algorithms are based on the use of the Laplace transform, which allows us to reduce the resolution of differential equations with constant coefficients to manipulations with rational functions. The complexity results are summarized in the following theorem.

Theorem 2. *Let* N *and* r *be two positive integers and let* $\mathbb{K}$ *be a field of characteristic zero or at least* N*. Then, for differential equations and systems with constant coefficients:*

(a) *problem* **i** *can be solved using* $\mathcal{O}(\mathsf{M}(r)(r+N))$ *operations in* $\mathbb{K}$*;*
(b) *problem* **ii** *can be solved using* $\mathcal{O}(\mathsf{M}(r)(1+N/r))$ *operations in* $\mathbb{K}$*;*
(c) *problem* **I** *can be solved using* $\mathcal{O}\left(r^{\omega+1}\log r + r\mathsf{M}(r)N\right)$ *operations in* $\mathbb{K}$*;*
(d) *problem* **II** *can be solved using* $\mathcal{O}(r^\omega \log r + \mathsf{M}(r)N)$ *operations in* $\mathbb{K}$*.*

In the case of polynomial coefficients, we exploit the linear recurrence satisfied by the coefficients of solutions. In Table 1, we gather the complexity estimates corresponding to the best known solutions for each of the four problems **i**, **ii**, **I**, and **II** in the general case, as well as in the above mentioned special cases. The algorithms are described in Section 4. In the polynomial coefficients case, these results are well known. In the other cases, to the best of our knowledge, the results improve upon existing algorithms.

1.4. **Non-linear Systems.** As an important consequence of Theorem 1, we improve the known complexity results for the more general problem of solving *non-linear* systems of differential equations. To do so, we use a classical reduction technique from the non-linear to the linear case, see for instance [34, Section 25] and [8, Section 5.2]. For simplicity, we only consider non-linear systems of first order. There is no loss of generality in doing so, more general cases can be reduced to that one by adding new unknowns and possibly differentiating once. The following result generalizes [8, Theorem 5.1]. If $F = (F_1, \ldots, F_r)$ is a differentiable function bearing on r variables $y_1, \ldots, y_r$, we use the notation $\mathbf{Jac}(F)$ for the Jacobian matrix $(\partial F_i/\partial y_j)_{1\le i,j\le r}$.

Theorem 3. *Let* N*,* r *be in* $\mathbb{N}$*, let* $\mathbb{K}$ *be a field of characteristic zero or at least* N *and let* φ *denote* $(\varphi_1, \ldots, \varphi_r)$*, where* $\varphi_i(t, y)$ *are multivariate power series in* $\mathbb{K}[[t, y_1, \ldots, y_r]]$*.*

Let $\mathsf{L} : \mathbb{N} \to \mathbb{N}$ *be such that for all* $s(t)$ *in* $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ *and for all* n *in* $\mathbb{N}$*, the first* n *terms of* $\varphi(t, s(t))$ *and of* $\mathbf{Jac}(\varphi)(t, s(t))$ *can be computed in* $\mathsf{L}(n)$ *operations in* $\mathbb{K}$*. Suppose in addition that the function* $n \mapsto \mathsf{L}(n)/n$ *is increasing. Given initial conditions* v *in* $\mathcal{M}_{r\times 1}(\mathbb{K})$*, if the differential system*

$$y' = \varphi(t, y), \qquad y(0) = v,$$

admits a solution in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$*, then the first* N *terms of such a solution* $y(t)$ *can be computed in* $\mathcal{O}\left(\mathsf{L}(N) + \min(\mathsf{MM}(r, N), r^2\mathsf{M}(N)\log N)\right)$ *operations in* $\mathbb{K}$*.*

Werschulz [41, Theorem 3.2] gave an algorithm solving the same problem using the integral Volterra-type equation technique described in [34, pp. 172–173]. With our notation, his algorithm uses $\mathcal{O}\left(\mathsf{L}(N) + r^2 N\,\mathsf{M}(N))\right)$ operations in $\mathbb{K}$ to compute a solution at precision N. Thus, our algorithm is an improvement for cases where $\mathsf{L}(N)$ is known to be subquadratic with respect to N.

The best known algorithms for power series composition in $r \geq 2$ variables require, at least on "generic" entries, a number $\mathsf{L}(n) = \mathcal{O}(n^{r-1}\mathsf{M}(n))$ of operations in $\mathbb{K}$ to compute the first n coefficients of the composition [7, Section 3]. This complexity is nearly optimal with respect to the size of a generic input. By contrast, in the univariate case, the best known result [8, Th. 2.2] is $\mathsf{L}(n) = \mathcal{O}(\sqrt{n\log n}\,\mathsf{M}(n))$. For special entries, however, better results can be obtained, already in the univariate case: exponentials, logarithms, powers of univariate power series can be computed [6, Section 13] in $\mathsf{L}(n) = \mathcal{O}(\mathsf{M}(n))$. As a consequence, if φ is an r-variate sparse polynomial with m monomials of *any* degree, then $\mathsf{L}(n) = \mathcal{O}(mr\,\mathsf{M}(n))$.

Another important class of systems with such a subquadratic $\mathsf{L}(N)$ is provided by *rational systems*, where each φ_i is in $\mathbb{K}(y_1, \ldots, y_r)$. Supposing that the complexity of evaluation of φ is bounded by L (i.e., for any point z in $\mathbb{K}^r$ at which φ is well-defined, the value $\varphi(z)$ can be computed using at most L operations in $\mathbb{K}$), then, the Baur-Strassen theorem [2] implies that the complexity of evaluation of the Jacobian $\mathbf{Jac}(\varphi)$ is bounded by $5L$, and therefore, we can take $\mathsf{L}(n) = \mathsf{M}(n)L$ in the statement of Theorem 3.

1.5. **Basic Complexity Notation.** Our algorithms ultimately use, as a basic operation, multiplication of matrices with entries that are polynomials (or truncated power series). Thus, to estimate their complexities in a unified manner, we use a function $\mathsf{MM} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that any two $r \times r$ matrices with polynomial entries in $\mathbb{K}[t]$ of degree less than d can be multiplied using $\mathsf{MM}(r, d)$ operations in $\mathbb{K}$. In particular, $\mathsf{MM}(1, d)$ represents the number of base field operations required to multiply two polynomials of degree less than d, while $\mathsf{MM}(r, 1)$ is the arithmetic cost of scalar $r \times r$ matrix multiplication. For simplicity, we denote $\mathsf{MM}(1, d)$ by $\mathsf{M}(d)$ and we have $\mathsf{MM}(r, 1) = \mathcal{O}(r^\omega)$, where $2 \leq \omega \leq 3$ is the so-called *exponent of the matrix multiplication*, see, e.g., [9] and [14].

Using the algorithms of [36, 10], one can take $\mathsf{M}(d)$ in $\mathcal{O}(d\log d\log\log d)$; over fields supporting FFT, one can take $\mathsf{M}(d)$ in $\mathcal{O}(d\log d)$. By [10] we can always choose $\mathsf{MM}(r, d)$ in $\mathcal{O}(r^\omega\,\mathsf{M}(d))$, but better estimates are known in important particular cases. For instance, over fields of characteristic 0 or larger than $2d$, we have $\mathsf{MM}(r, d) = \mathcal{O}(r^\omega d + r^2\,\mathsf{M}(d))$, see [5, Th. 4]. To simplify the complexity analyses of our algorithms, we suppose that the multiplication cost function MM satisfies the following standard growth hypotheses for all integers d_1, d_2 and r:

$$\mathsf{MM}(r, d_1 d_2) \leq d_1^2\mathsf{MM}(r, d_2) \qquad \text{and} \qquad \frac{\mathsf{MM}(r, d_1)}{d_1} \leq \frac{\mathsf{MM}(r, d_2)}{d_2} \quad \text{if } d_1 \leq d_2. \tag{2}$$

In particular, Equation (2) implies the inequalities

$$\begin{aligned} \mathsf{MM}(r, 2^\kappa) + \mathsf{MM}(r, 2^{\kappa-1}) + M(r, 2^{\kappa-2}) + \cdots + \mathsf{MM}(r, 1) &\leq 2\mathsf{MM}(r, 2^\kappa), \\ \mathsf{M}(2^\kappa) + 2\mathsf{M}(2^{\kappa-1}) + 4\mathsf{M}(2^{\kappa-2}) + \cdots + 2^\kappa\mathsf{M}(1) &\leq (\kappa+1)\mathsf{M}(2^\kappa). \end{aligned} \tag{3}$$

These inequalities are crucial to prove the estimates in Theorem 1 and Theorem 3. Note also that when the available multiplication algorithm is slower than quasi-linear (e.g., Karatsuba or naive

multiplication), then in the second inequality, the factor $(\kappa+1)$ can be replaced by a constant and thus the estimates $\mathsf{M}(N)\log N$ in our complexities become $\mathsf{M}(N)$ in those cases.

1.6. **Notation for Truncation.** It is recurrent in algorithms to split a polynomial into a lower and a higher part. To this end, the following notation proves convenient. Given a polynomial f, the remainder and quotient of its Euclidean division by t^k are respectively denoted $\lceil f \rceil^k$ and $\lfloor f \rfloor_k$. Another occasional operation consists in taking a middle part out of a polynomial. To this end, we let $[f]_k^l$ denote $\left\lfloor \lceil f \rceil^l \right\rfloor_k$. Furthermore, we shall write $f = g \mod t^k$ when two polynomials or series f and g agree up to degree $k-1$ included. To get a nice behaviour of integration with respect to truncation orders, all primitives of series are chosen with zero as their constant coefficient.

2. Newton Iteration for Systems of Linear Differential Equations

Let $Y'(t) = A(t)Y(t) + B(t)$ be a linear differential system, where $A(t)$ and $B(t)$ are $r \times r$ matrices with coefficients in $\mathbb{K}[[t]]$. Given an invertible scalar matrix Y_0, an integer $N \geq 1$, and the expansions of A and B up to precision N, we show in this section how to compute efficiently the power series expansion at precision N of the unique solution of the Cauchy problem

$$Y'(t) = A(t)Y(t) + B(t) \quad \text{and} \quad Y(0) = Y_0.$$

This enables us to answer problems **I** and **i**, the latter being a particular case of the former (through the application to a companion matrix).

2.1. **Homogeneous Case.** First, we design a Newton-type iteration to solve the homogeneous system $Y' = A(t)Y$. The classical Newton iteration to solve an equation $\phi(y) = 0$ is $Y_{\kappa+1} = Y_\kappa - U_\kappa$, where U_κ is a solution of the linearized equation $D\phi|_{Y_\kappa} \cdot U = \phi(Y_\kappa)$ and $D\phi|_{Y_\kappa}$ is the differential of ϕ at Y_κ. We apply this idea to the map $\phi : Y \mapsto Y' - AY$. Since ϕ is linear, it is its own differential and the equation for U becomes

$$U' - AU = Y'_\kappa - AY_\kappa.$$

Taking into account the proper orders of truncation and using Lagrange's method of variation of parameters [24, 20], we are thus led to the iteration

$$\begin{cases} Y_{\kappa+1} &= Y_\kappa - \lceil U_\kappa \rceil^{2^{\kappa+1}}, \\ U_\kappa &= Y_\kappa \int \left\lceil Y_\kappa^{-1} \right\rceil^{2^{\kappa+1}} \left(Y'_\kappa - \lceil A \rceil^{2^{\kappa+1}} Y_\kappa \right). \end{cases}$$

Thus we need to compute (approximations of) the solution Y and its inverse simultaneously. Now, a well-known Newton iteration for the inverse Z of Y is

$$Z_{\kappa+1} = \lceil Z_\kappa + Z_\kappa(I_r - YZ_\kappa) \rceil^{2^{\kappa+1}}. \tag{4}$$

It was introduced by Schulz [37] in the case of real matrices; its version for matrices of power series is given for instance in [28].

Putting together these considerations, we arrive at the algorithm SolveHomDiffSys in Figure 1, whose correctness easily follows from Lemma 1 below. Remark that in the scalar case ($r = 1$) algorithm SolveHomDiffSys coincides with the algorithm for power series exponential proposed by Hanrot and Zimmermann [18]; see also [3]. In the case $r > 1$, ours is a nontrivial generalization of the latter. Because it takes primitives of series at precision N, algorithm SolveHomDiffSys requires that the elements $2, 3, \ldots, N-1$ be invertible in $\mathbb{K}$. Its complexity C satisfies the recurrence $\mathsf{C}(m) = \mathsf{C}(m/2) + \mathcal{O}(\mathsf{M}(r, m))$, which implies — using the growth hypotheses on M — that $\mathsf{C}(N) = \mathcal{O}(\mathsf{M}(r, N))$. This proves the first assertion of Theorem 1.

SolveHomDiffSys(A, N, Y_0)

Input: $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$.

Output: $Y = \sum_{i=0}^{N-1} Y_i t^i$ in $\mathcal{M}_{r\times r}(\mathbb{K})[t]$ such that $Y' = AY \mod t^{N-1}$, and $Z = Y^{-1} \mod t^{N/2}$.

$Y \leftarrow (I_r + tA_0)Y_0$
$Z \leftarrow Y_0^{-1}$
$m \leftarrow 2$
while $m \leq N/2$ do
  $Z \leftarrow Z + \lceil Z(I_r - YZ)\rceil^m$
  $Y \leftarrow Y - \left\lceil Y\left(\int Z(Y' - \lceil A\rceil^{2m-1} Y)\right)\right\rceil^{2m}$
  $m \leftarrow 2m$
return Y, Z

FIGURE 1. Solving the Cauchy problem $Y' = A(t)Y$, $Y(0) = Y_0$ by Newton iteration.

Lemma 1. *Let m be an even integer. Suppose that $Y_{(0)}, Z_{(0)}$ in $\mathcal{M}_{r\times r}(\mathbb{K}[t])$ satisfy*

$$I_r - Y_{(0)}Z_{(0)} = 0 \mod t^{m/2} \quad \text{and} \quad Y'_{(0)} - AY_{(0)} = 0 \mod t^{m-1},$$

and that they are of degree less than $m/2$ and m, respectively. Define

$$Z := \left\lceil Z_{(0)}\left(2I_r - Y_{(0)}Z_{(0)}\right)\right\rceil^m \quad \text{and} \quad Y := \left\lceil Y_{(0)}\left(I_r - \int Z(Y'_{(0)} - AY_{(0)})\right)\right\rceil^{2m}.$$

Then Y and Z satisfy the equations

$$I_r - YZ = 0 \mod t^m \quad \text{and} \quad Y' - AY = 0 \mod t^{2m-1}. \tag{5}$$

PROOF. Using the definitions of Y and Z, it follows that

$$I_r - YZ = (I_r - Y_{(0)}Z_{(0)})^2 - (Y - Y_{(0)})Z_{(0)}(2I_r - Y_{(0)}Z_{(0)}) \mod t^m.$$

Since by hypothesis $I_r - Y_{(0)}Z_{(0)}$ and $Y - Y_{(0)}$ are zero modulo $t^{m/2}$, the right-hand side is zero modulo t^m and this establishes the first formula in Equation (5). Similarly, write $Q = \int Z(Y'_{(0)} - AY_{(0)})$ and observe $Q = 0 \mod t^m$ to get the equality

$$Y' - AY = (I - YZ)(Y'_{(0)} - AY_{(0)}) - (Y'_{(0)} - AY_{(0)})Q \mod t^{2m-1}.$$

Now, $Y'_{(0)} - AY_{(0)} = 0 \mod t^{m-1}$, while Q and $I_r - YZ$ are zero modulo t^m and therefore the right-hand side of the last equation is zero modulo t^{2m-1}, proving the last part of the lemma. □

2.2. **General Case.** We want to solve the equation $Y' = AY + B$, where B is an $r \times r$ matrix with coefficients in $\mathbb{K}[[t]]$. Suppose that we have already computed the solution $\widetilde{Y}$ of the associate homogeneous equation $\widetilde{Y}' = A\widetilde{Y}$, together with its inverse $\widetilde{Z}$. Then, by the method of variation of parameters, $Y_{(1)} = \widetilde{Y}\int \widetilde{Z}B$ is a particular solution of the inhomogeneous problem, thus the general solution has the form $Y = Y_{(1)} + \widetilde{Y}$.

Now, to compute the particular solution $Y_{(1)}$ at precision N, we need to know both $\widetilde{Y}$ and $\widetilde{Z}$ at the same precision N. To do this, we first apply the algorithm for the homogeneous case and iterate (4) once. The resulting algorithm is encapsulated in Figure 2.

SolveInhomDiffSys(A, B, N, Y_0)

Input: $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$, $B_0, \ldots, B_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $B(t) = \sum B_i t^i$.

Output: $Y_1, \ldots, Y_{N-1}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$ such that $Y = Y_0 + \sum Y_i t^i$ satisfies $Y' = AY + B \mod t^{N-1}$.

$\widetilde{Y}, \widetilde{Z} \leftarrow$ SolveHomDiffSys(A, N, Y_0)

$\widetilde{Z} \leftarrow \widetilde{Z} + \left\lceil \widetilde{Z}(I_r - \widetilde{Y}\widetilde{Z}) \right\rceil^N$

$Y \leftarrow \left\lceil \widetilde{Y} \int (\widetilde{Z}B) \right\rceil^N$

$Y \leftarrow Y + \widetilde{Y}$

return Y

FIGURE 2. Solving the Cauchy problem $Y' = AY + B$, $Y(0) = Y_0$ by Newton iteration.

3. Divide-and-conquer Algorithm

We now give our second algorithm, which allows us to solve problems **ii** and **II** and to finish the proof of Theorem 1. Before entering a detailed presentation, let us briefly sketch the main idea in the particular case of a homogeneous differential equation $\mathcal{L}y = 0$, where $\mathcal{L}$ is a linear differential operator in $\delta = t\frac{d}{dt}$ with coefficients in $\mathbb{K}[[t]]$. (The introduction of δ is only for pedagogical reasons.) The starting remark is that if a power series y is written as $y_0 + t^m y_1$, then $\mathcal{L}(\delta)y = \mathcal{L}(\delta)y_0 + t^m\mathcal{L}(\delta + m)y_1$. Thus, to compute a solution y of $\mathcal{L}(\delta)y = 0 \mod t^{2m}$, it suffices to determine the lower part of y as a solution of $\mathcal{L}(\delta)y_0 = 0 \mod t^m$, and then to compute the higher part y_1, as a solution of the inhomogeneous equation $\mathcal{L}(\delta + m)y_1 = -R \mod t^m$, where the rest R is computed so that $\mathcal{L}(\delta)y_0 = t^m R \mod t^{2m}$.

Our algorithm DivideAndConquer makes a recursive use of this idea. Since, during the recursions, we are naturally led to treat inhomogeneous equations of a slightly more general form than that of **II** we introduce the notation $\mathcal{E}(s, p, m)$ for the vector equation

$$ty' + (pI_r - tA)y = s \mod t^m.$$

The algorithm is described in Figure 3. Choosing $p = 0$ and $s(t) = tb(t)$ we retrieve the equation of problem **II**. Our algorithm Solve to solve problem **II** is thus a specialization of DivideAndConquer, defined by making Solve(A, b, N, v) simply call DivideAndConquer$(tA, tb, 0, N, v)$. Its correctness relies on the following immediate lemma.

Lemma 2. *Let A in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$, s in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$, and let p, d in $\mathbb{N}$. Decompose $\lceil s \rceil^m$ into a sum $s_0 + t^d s_1$. Suppose that y_0 in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ satisfies the equation $\mathcal{E}(s_0, p, d)$, set R to be*

$$\left\lceil (ty_0' + (pI_r - tA)y_0 - s_0)/t^d \right\rceil^{m-d},$$

and let y_1 in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ be a solution of the equation $\mathcal{E}(s_1 - R, p + d, m - d)$. Then the sum $y := y_0 + t^d y_1$ is a solution of the equation $\mathcal{E}(s, p, m)$.

The only divisions performed along our algorithm Solve are by $1, \ldots, N - 1$. As a consequence of this remark and of the previous lemma, we deduce the complexity estimates in the proposition below; for a general matrix A, this proves point (c) in Theorem 1, while the particular case when A is companion proves point (b).

DivideAndConquer(A, s, p, m, v)

Input: $A_0, \ldots, A_{m-1}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$, $s_0, \ldots, s_{m-1}, v$ in $\mathcal{M}_{r\times 1}(\mathbb{K})$, $s = \sum s_i t^i$, p in $\mathbb{K}$.

Output: $y = \sum_{i=0}^{N-1} y_i t^i$ in $\mathcal{M}_{r\times 1}(\mathbb{K})[t]$ such that $ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$.

```
If m = 1 then
   if p = 0 then
      return v
   else return p^{-1} s(0)
end if
d ← ⌊m/2⌋
s ← ⌈s⌉^d
y_0 ← DivideAndConquer(A, s, p, d, v)
R ← [s − t y_0' − (pI_r − tA) y_0]_d^m
y_1 ← DivideAndConquer(A, R, p + d, m − d, v)
return y_0 + t^d y_1
```

FIGURE 3. Solving $ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$, by divide-and-conquer.

Proposition 1. *Given the first m terms of the entries of $A \in \mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ and of $s \in \mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$, given $v \in \mathcal{M}_{r\times 1}(\mathbb{K})$, algorithm* DivideAndConquer(A, s, p, m, v) *computes a solution of the linear differential system $ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$, using $\mathcal{O}(r^2\,\mathsf{M}(m)\log m)$ operations in $\mathbb{K}$. If A is a companion matrix, the cost reduces to $\mathsf{C}(m) = \mathcal{O}(r\,\mathsf{M}(m)\log m)$.*

PROOF. The correctness of the algorithm follows from the previous Lemma. The cost $\mathsf{C}(m)$ of the algorithm satisfies the recurrence

$$\mathsf{C}(m) = \mathsf{C}(\lfloor m/2\rfloor) + \mathsf{C}(\lceil m/2\rceil) + r^2\,\mathsf{M}(m) + \mathcal{O}(rm),$$

where the term $r^2\,\mathsf{M}(m)$ comes from the application of A to y_0 used to compute the rest R. From this recurrence, it is easy to infer that $\mathsf{C}(m) = \mathcal{O}(r^2\,\mathsf{M}(m)\log m)$. Finally, when A is a companion matrix, the vector R can be computed in time $O(r\,\mathsf{M}(m))$, which implies that in this case $\mathsf{C}(m) = \mathcal{O}(r\,\mathsf{M}(m)\log m)$. □

4. Faster Algorithms for Special Coefficients

4.1. **Constant Coefficients.** Let A be a constant $r \times r$ matrix and let v be a vector of initial conditions. Given $N \geq 1$, we want to compute the first N coefficients of the series expansion of a solution y in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ of $y' = Ay$, with $y(0) = v$. In this setting, many various algorithms have been proposed to solve problems **i**, **ii**, **I**, and **II**, see for instance [32, 33, 21, 12, 29, 25, 26, 16, 17, 19, 30, 27]. Again, the most naive algorithm is based on the method of undetermined coefficients. On the other hand, most books on differential equations, see, e.g., [20, 11, 1] recommend to simplify the calculations using the Jordan form of matrices. The main drawback of that approach is that computations are done over the algebraic closure of the base field $\mathbb{K}$. The best complexity result known to us is given in [27] and it is quadratic in r.

We concentrate first on problems **ii** and **II** (computing a single solution for a single equation, or a first-order system). Our algorithm for problem **II** uses $\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$ operations in $\mathbb{K}$ for a general constant matrix A and only $\mathcal{O}(N\mathsf{M}(r)/r)$ operations in $\mathbb{K}$ in the case where A is a companion matrix (problem **ii**). Despite the simplicity of the solution, this is, to the best of our knowledge, a new result.

In order to compute $y_N = \sum_{i=0}^{N} A^i v t^i/i!$, we first compute its Laplace transform $z_N = \sum_{i=0}^{N} A^i v t^i$: indeed, one can switch from y_N to z_N using only $\mathcal{O}(Nr)$ operations in $\mathbb{K}$. The vector z_N is the truncation at order $N+1$ of $z = \sum_{i\geq 0} A^i v t^i = (I - tA)^{-1} v$. As a byproduct of a more difficult question, [38, Prop. 10] shows that z_N can be computed using $\mathcal{O}(Nr^{\omega-1})$ operations in $\mathbb{K}$. We propose a solution of better complexity.

By Cramer's rule, z is a vector of rational functions $z_i(t)$, of degree at most r. The idea is to first compute z as a rational function, and then to deduce its expansion modulo t^{N+1}. The first part of the algorithm does not depend on N and thus it can be seen as a precomputation. For instance, one can use [38, Corollary 12], to compute z in complexity $\mathcal{O}(r^\omega \log r)$. In the second step of the algorithm, we have to expand r rational functions of degree at most r at precision N. Each such expansion can be performed using $\mathcal{O}(N\mathsf{M}(r)/r)$ operations in $\mathbb{K}$, see, e.g., the proof of [4, Prop. 1]. The total cost of the algorithm is thus $\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$. We give below a simplified variant with same complexity, avoiding the use of the algorithm in [38] for the precomputation step and relying instead on a technique which is classical in the computation of minimal polynomials [9].

(1) Compute the vectors $v, Av, A^2v, A^3v, \ldots, A^{2r}v$ in $\mathcal{O}(r^\omega \log r)$, as follows:
for κ from 1 to $1 + \log r$ do
  (a) compute A^{2^κ}
  (b) compute $A^{2^\kappa} \times [v|Av|\cdots|A^{2^\kappa-1}v]$, thus getting $[A^{2^\kappa}v|A^{2^\kappa+1}v|\cdots|A^{2^{\kappa+1}-1}v]$

(2) For each $j = 1, \ldots, r$:
  (a) recover the rational fraction whose series expansion is $\sum (A^i v)_j t^i$ by Padé approximation in $\mathcal{O}(\mathsf{M}(r)\log r)$ operations
  (b) compute its expansion up to precision t^{N+1} in $\mathcal{O}(N\,\mathsf{M}(r)/r)$ operations
  (c) recover the expansion of y from that of z, using $\mathcal{O}(N)$ operations.

This yields the announced total cost of $\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$ operations for problem **II**.

We now turn to the estimation of the cost for problems **i** and **I** (bases of solutions). In the case of equations with constant coefficients, we use the Laplace transform again. If $y = \sum_{i\geq 0} y_i t^i$ is a solution of an order r equation with constant coefficients, then the sequence $(z_i) = (i!y_i)$ is generated by a linear recurrence with constant coefficients. Hence, the first terms $z_1, \ldots, z_N$ can be computed in $O(N\mathsf{M}(r)/r)$ operations, using again the algorithm described in [4, Prop. 1]. For problem **I**, the exponent $\omega + 1$ in the cost of the precomputation can be reduced to ω by a very different approach; we cannot give the details here for space limitation.

4.2. **Polynomial Coefficients.** If the coefficients in one of the problems **i, ii, I**, and **II** are polynomials in $\mathbb{K}[t]$ of degree at most d, using the linear recurrence of order d satisfied by the coefficients of the solution seemingly yields the lowest possible complexity. Consider for instance problem **II**. Plugging $A = \sum_{i=0}^{d} t^i A_i$, $b = \sum_{i=0}^{d} t^i b_i$, and $y = \sum_{i\geq 0}^{d} t^i y_i$ in the equation $y' = Ay + b$, we arrive at the following recurrence

$$y_{k+d+1} = (d+k+1)^{-1}(A_d y_k + A_{d-1} y_{k+1} + \cdots + A_0 y_{k+d} + b_{k+d}), \quad \text{for all } k \geq -d.$$

Thus, to compute $y_0, \ldots, y_N$, we need to perform Nd matrix-vector products; this is done using $\mathcal{O}(dNr^2)$ operations in $\mathbb{K}$. A similar analysis implies the other complexity estimates in the third column of Table 1.

5. Non-linear Systems of Differential Equations

Let $\varphi(t, y) = (\varphi_1(t, y), \ldots, \varphi_r(t, y))$, where each φ_i is a power series in $\mathbb{K}[[t, y_1, \ldots, y_r]]$. We consider the first-order non-linear system in y

$$(\mathcal{N}) \qquad y_1'(t) = \varphi_1(t, y_1(t), \ldots, y_r(t)), \quad \ldots, \quad y_r'(t) = \varphi_r(t, y_1(t), \ldots, y_r(t)).$$

To solve $(\mathcal{N})$, we use the classical technique of *linearization*. The idea is to attach, to an *approximate* solution y_0 of $(\mathcal{N})$, a *tangent* system in the new unknown z,

$$(\mathcal{T}, y_0) \qquad z' = \mathbf{Jac}(\varphi)(y_0)z - y_0' + \varphi(y_0),$$

which is linear and whose solutions serve to obtain a better approximation of a true solution of $(\mathcal{N})$. Indeed, let us denote by $(\mathcal{N}_m), (\mathcal{T}_m)$ the systems $(\mathcal{N}), (\mathcal{T})$ where all the equalities are taken modulo t^m. Taylor's formula states that the expansion $\varphi(y+z) - \varphi(y) - \mathbf{Jac}(\varphi)(y)z$ is equal to 0 modulo z^2. It is a simple matter to check that if y is a solution of $(\mathcal{N}_m)$ and if z is a solution of $(\mathcal{T}_{2m}, y)$, then $y+z$ is a solution of $(\mathcal{N}_{2m})$. This justifies the correctness of Algorithm SolveNonLinearSys.

To analyze the complexity of this algorithm, it suffices to remark that for each integer κ between 1 and $\lfloor \log N \rfloor$, one has to compute one solution of a linear inhomogeneous first-order system at precision 2^κ and to evaluate φ and its Jacobian on a series at the same precision. This concludes the proof of Theorem 3.

SolveNonLinearSys(ϕ, v)

Input: N in $\mathbb{N}$, $\varphi(t, y)$ in $\mathbb{K}[[t, y_1, \ldots, y_r]]^r$, v in $\mathbb{K}^r$

Output: first N terms of a $y(t)$ in $\mathbb{K}[[t]]$ such that $y(t)' = \varphi(t, y(t)) \mod t^N$ and $y(0) = v$.

$m \leftarrow 1$
$y \leftarrow v$
while $m \leq N/2$ do
    $A \leftarrow \lceil \mathbf{Jac}(\varphi)(y) \rceil^{2m}$
    $b \leftarrow \lceil \varphi(y) - y' \rceil^{2m}$
    $z \leftarrow$ Solve$(A, b, 2m, 0)$
    $y \leftarrow y + z$
    $m \leftarrow 2m$
return y

FIGURE 4. Solving the non-linear differential system $y' = \varphi(t, y),\ y(0) = v$.

6. Implementation and Timings

We implemented our algorithms SolveDiffHomSys and Solve in Magma [31] and ran the programs on an Athlon processor at 2.2 GHz with 2 GB of memory.[1] We used Magma's built-in polynomial arithmetic (using successively naive, Karatsuba and FFT multiplication algorithms), as well as Magma's scalar matrix multiplication (of cubic complexity in the ranges of our interest). We give three tables of timings. First, we compare in Figure 5 the performances of our algorithm SolveDiffHomSys with that of the naive quadratic algorithm, for computing a basis of (truncated power series) solutions of a homogeneous system. The order of the system varies from 2 to 16, while the precision required for the solution varies from 256 to 4096; the base field is $\mathbb{Z}/p\mathbb{Z}$, where p is a 32-bit prime.

Then we display in Figure 6 and Figure 7 the timings obtained respectively with algorithm SolveDiffHomSys and with the algorithm for polynomial matrix multiplication PolyMatMul that was

[1] All the computations have been done on the machines of the MEDICIS ressource center http://medicis.polytechnique.fr.

$N \setminus r$	2	4	8	16
256	0.02 vs. 2.09	0.08 vs. 6.11	0.44 vs. 28.16	2.859 vs. 168.96
512	0.04 vs. 8.12	0.17 vs. 25.35	0.989 vs. 113.65	6.41 vs. 688.52
1024	0.08 vs. 32.18	0.39 vs. 104.26	2.30 vs. 484.16	15 vs. 2795.71
2048	0.18 vs. 128.48	0.94 vs. 424.65	5.54 vs. 2025.68	36.62 vs. $>$ 3hours *
4096	0.42 vs. 503.6	2.26 vs. 1686.42	13.69 vs. 8348.03	92.11 vs. $>$ 1/2 day*

FIGURE 5. Computation of a basis of a linear homogeneous system with r equations, at precision N: comparison of timings (in seconds) between algorithm SolveDiffHomSys and the naive algorithm. Entries marked with a '$\star$' are estimated timings.

used as a primitive of SolveDiffHomSys. The similar shapes of the two surfaces indicate that the complexity prediction of point (a) in Theorem 1 is well respected in our implementation: SolveDiffHomSys uses a constant number (between 4 and 5) of polynomial multiplications; note that the abrupt jumps at powers of 2 reflect the performance of Magma's FFT implementation of polynomial arithmetic.

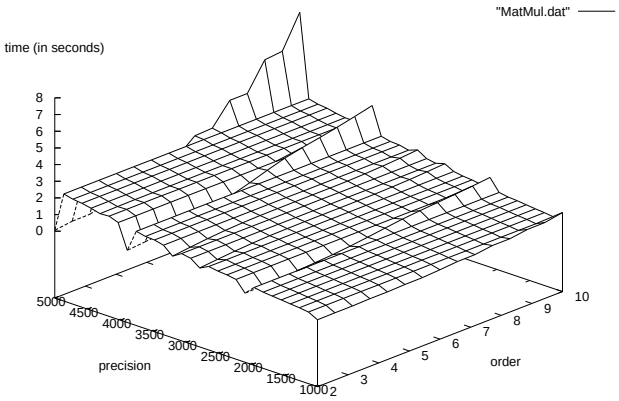

FIGURE 6. Timings of algorithm PolyMatMul.

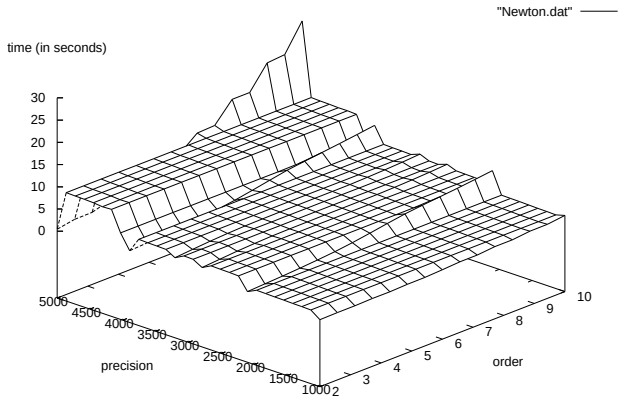

FIGURE 7. Timings of algorithm SolveDiffHomSys.

In Figure 8 we give the timings for the computation of one solution of a linear differential equation of order 2, 4, and 8, respectively, using our algorithm Solve in Section 3. Again, the shape of the three curves experimentally confirms the nearly linear behaviour established in point (b) of Theorem 1, both in the precision N and in the order r of the complexity of algorithm Solve. Finally, Figure 9 displays the three curves from Figure 8 together with the timings curve for the naive quadratic algorithm computing one solution of a linear differential equation of order 2. The conclusion is that our algorithm Solve becomes very early superior to the quadratic one.

We also implemented our algorithms of Section 4.1 for the special case of constant coefficients. For reasons of space limitation, we only provide a few experimental results for problem **II**. Over the same finite field, we computed: a solution of a linear system with $r = 8$ at precision $N \approx 10^6$ in 24.53s; one at doubled precision in doubled time 49.05s; one for doubled order $r = 16$ in doubled time 49.79s.

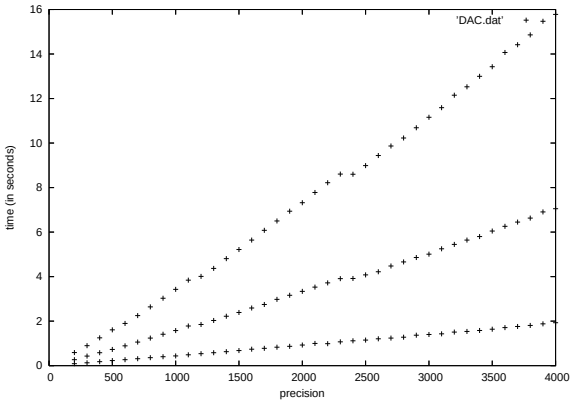

FIGURE 8. Timings of algorithm Solve for equations of orders 2, 4, and 8.

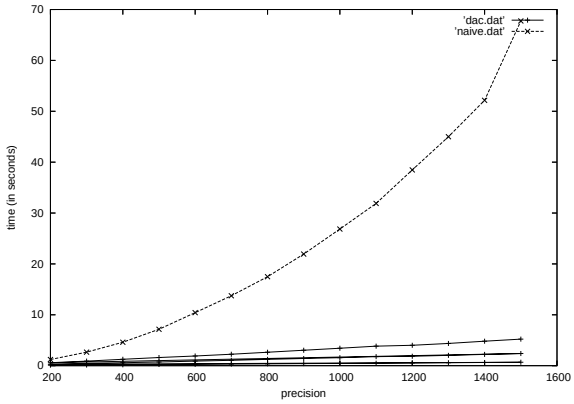

FIGURE 9. Same, compared to the naive algorithm for a second-order equation.

REFERENCES

[1] V. I. Arnol′d. *Ordinary differential equations*. Springer Textbook. Springer-Verlag, Berlin, 1992. Translated from the third Russian edition by Roger Cooke.

[2] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–320, 1983.

[3] D. J. Bernstein. Removing redundancy in high-precision Newton iteration, 2000. Available on-line at http://cr.yp.to/fastnewton.html.

[4] A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *Journal of Symbolic Computation*, 41(1):1–29, January 2006.

[5] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, August 2005. Festschrift for the 70th Birthday of Arnold Schönhage.

[6] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity*, pages 151–176. Academic Press, New York, 1976. Proceedings of a Symposium held at Carnegie-Mellon University, Pittsburgh, Pa., 1975.

[7] R. P. Brent and H. T. Kung. Fast algorithms for composition and reversion of multivariate power series. In *Proceedings of the Conference on Theoretical Computer Science, University of Waterloo, Waterloo, Ontario Canada, August 1977*, pages 149 – 158, 1977.

[8] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25(4):581–595, 1978.

[9] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren Math. Wiss.* Springer–Verlag, 1997.

[10] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[11] E. A. Coddington. *An introduction to ordinary differential equations*. Prentice-Hall Mathematics Series. Prentice-Hall Inc., Englewood Cliffs, N.J., 1961.

[12] E. P. Fulmer. Computation of the matrix exponential. *Amer. Math. Monthly*, 82(2):156–159, 1975.

[13] J. von zur Gathen and J. Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In *ISSAC'97*, pages 40–47. ACM Press, 1997.

[14] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.
[15] K. O. Geddes. Convergence behavior of the Newton iteration for first order differential equations. In *EUROSAM '79: Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, pages 189–199, London, UK, 1979. Springer-Verlag.
[16] C. Q. Gu. A new Padé approximation algorithm for matrix exponentials. *Acta Automat. Sinica*, 25(1):94–99, 1999.
[17] C. Q. Gu. Continued fraction algorithm for matrix exponentials. *J. Shanghai Univ.*, 5(1):11–14, 2001.
[18] G. Hanrot and P. Zimmermann. Newton iteration revisited, 2002. Available on-line at http://www.loria.fr/~zimmerma/papers/fastnewton.ps.gz.
[19] W. A. Harris, Jr., J. P. Fillmore, and D. R. Smith. Matrix exponentials—another approach. *SIAM Rev.*, 43(4):694–706, 2001.
[20] E. L. Ince. *Ordinary Differential Equations*. New York: Dover Publications, 1956.
[21] R. B. Kirchner. An explicit formula for e^{At}. *Amer. Math. Monthly*, 74(10):1200–1204, 1967.
[22] D. E. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens (Nice, 1970), Tome 3*, pages 269–274. Gauthier-Villars, Paris, 1971.
[23] H. T. Kung and J. F. Traub. All algebraic functions can be computed fast. *Journal of the Association for Computing Machinery*, 25(2):245–260, 1978.
[24] J.-L. Lagrange. *Œuvres*. Editions Jacques Gabay, 1869.
[25] I. E. Leonard. The matrix exponential. *SIAM Rev.*, 38(3):507–512, 1996.
[26] E. Liz. A note on the matrix exponential. *SIAM Rev.*, 40(3):700–702, 1998.
[27] U. Luther and K. Rost. Matrix exponentials and inversion of confluent Vandermonde matrices. *Electron. Trans. Numer. Anal.*, 18:91–100, 2004.
[28] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille)*, volume 72 of *LNCS*, pages 65–73. Springer, Berlin, 1979.
[29] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.
[30] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.
[31] Computational Algebra Group (University of Sydney). The Magma computational algebra system. Available on-line at http://magma.maths.usyd.edu.au/.
[32] W. O. Pennell. A New Method for Determining a Series Solution of Linear Differential Equations with Constant or Variable Coefficients. *Amer. Math. Monthly*, 33(6):293–307, 1926.
[33] E. J. Putzer. Avoiding the Jordan canonical form in the discussion of linear systems with constant coefficients. *Amer. Math. Monthly*, 73(1):2–7, 1966.
[34] L. B. Rall. *Computational solution of nonlinear operator equations*. With an appendix by Ramon E. Moore. John Wiley & Sons Inc., New York, 1969.
[35] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Inform.*, 1:139–144, 1971.
[36] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
[37] G. Schulz. Iterative Berechnung der reziproken Matrix. *Z. Angew. Math. Mech.*, 13:57–59, 1933.
[38] A. Storjohann. High-order lifting. In *ISSAC'02*, pages 246–254. ACM, 2002.
[39] V. Strassen. The computational complexity of continued fractions. *SIAM Journal on Computing*, 12:1–27, 1983.
[40] J. van der Hoeven. Relax, but don't be too lazy. *J. Symbolic Comput.*, 34(6):479–542, 2002.

[41] A. G. Werschulz. Computational complexity of one-step methods for systems of differential equations. *Math. Comp.*, 34(149):155–174, 1980.